موضوع اصلی: تشخیص ارقام از روی تصاویر

هدف اساسی کد مورد نظر اینست که ارقام و حروف را از روی تصاویر تشخیص دهد. که این کـار یکی از حوزه های بینایی کامپیوتر (computer vision) میباشـد و بـه اصـطلاح بـه آن OCR یـا Optical character recognition می گویند. این هدف بسته به شرایط میتواند خیلی سخت و یا خیلی آسان شود.
این هدف را با توجه به موفقیت های چشمگیر شبکه های عصبی در بینایی ماشین، با استفاده از شبکه هـای عصـبی انجـام میدهیم.
به طور کلی روال اصلی اینست که ابتدا معماری شبکه عصبی خود را تعیین و سپس آن را پیـاده سـازی میکـنیم و بعـد از آن شروع به تمرین دادن شبکه خود با استفاده از داده های آموزشی میکنیم و خب هر چه قدر داده های آموزشی بیشـتر باشـند، شبکه قویتر میشود. با تمرین کافی، وزن هـا و بایـاس هـای شـبکه بـه مقـادیر بهینـه خـود میـل میکننـد. سـپس بـا اسـتفاده از مجموعه داده های تست از شبکه آزمون میگیریم و مشخص می کنیم که چقدر در کار خود موفق بوده ایم.

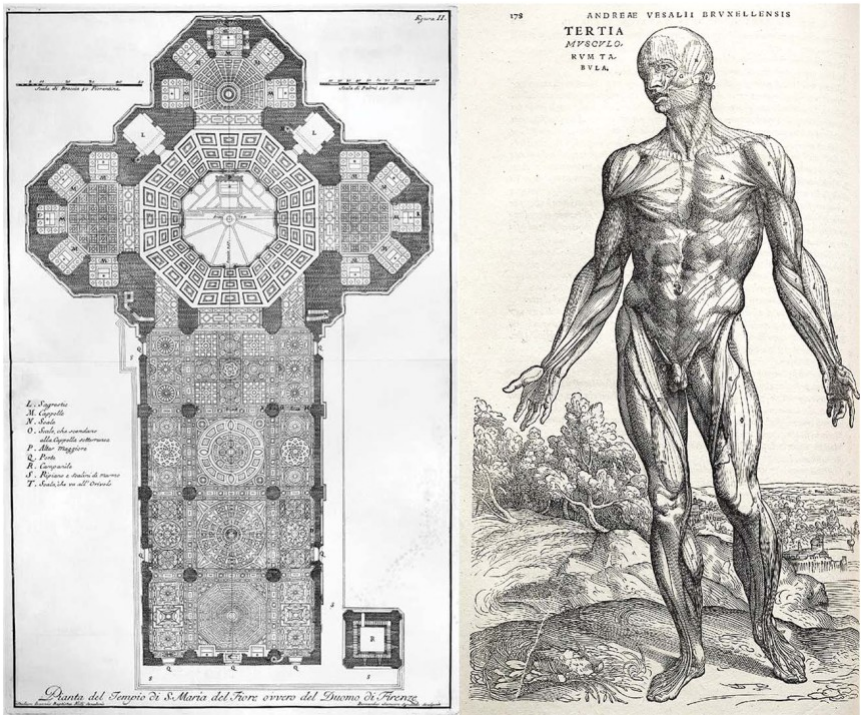
توضیحاتی در مورد معماری شبکه عصبی

خب در مقاله اصلی دانستیم که شبکه های عصبی cnn در حوزه بینایی ماشین توانـا هسـتند. ولی نکتـه در خور توجهی که اینجا وجود دارد اینست که چگونه چه معماری شبکه ای در نظر بگیریم، به عبارتی لایه ها چگونه به هم متصل شوند و ساختار چگونه باشند.
یک معماری مدل، مجموعـه انتخـاب هـایی اسـت کـه مـا انجـام میـدهیم بـه این صـورت کـه چـه لایـه ای انتخاب کنیم، چگونه آنها را تنظیم کنیم، بـه چـه ترتیـبی آنهـا را بچینیم. بـه این گزینـه هـا فضـای فرضـیه(Hypothesis space) مدل خود میگوییم، فضایی شامل توابع ممکنی که گرادیـان کاهشـی میتوانـد جسـتجو کند و توابعی که میتوانند بر اساس وزن های مدل، پارامتری شوند. به منظور یادگیری مـوثر از داده هـا، مـا نیاز داریم حدس ها و فرضیاتی در مورد چیزی که به دنبالش هستیم بزنیم.
معماری مدل معمولا تفاوت بین پیروزی و شکست است. اگر ما معماری بدی انتخاب کـنیم، مـدل مـا بـا معیار های غیربهینه گیر می افتد و پیشرفت چندانی حاصل نمیشود. در مقابل یک مدل معماری خوب به یادگیری شتاب خواهد بخشید و همچنین با تعداد کمـتری از داده بـه سـطح یـادگیری بیشـتری برسـد. یـک معمـاری مـدل خـوب ، انـدازه فضـای جسـتجو (search space) را کـاهش میدهـد یـا در غـیر این صـورت باعث راحت تر شدن همگرایی به یک نقطه خوب از فضای جستجو مشود. به زبان دیگر میتوان گفت کـه معماری مدل در مورد این است که مسئله را برای انجام گرادیان کاهشی ساده تر کنیم در نظر داشته باشـید که گرادیان کاهشی یک فرایند جستجوی احمقانه است (سرعت کمی دارد)، فلـذا تـا هـر چقـدر کـه امکـان دارد باید معماری را بهینه تر کرد.
معمـاری مـدل بیشـتر یـک هـنر اسـت تـا یـک علم. مهندسـان باتجربـه یـادگیری ماشـین ، **بـه طـور شـهودی** میتوانند در اولین تلاش های خود مدل های با کارایی بالا را کنار هم بچینند در حـالی کـه مبتـدیان معمـولا در حال مبارزه برای انتخاب مدل مناسب هستند و سختی های زیادی متحمل میشوند. توجه داشته باشید که از کلمه شهودی استفاده کردم، چرا که حقیقت این است که هیچ کس نمیتواند توضیح شفاف و دقیقی

ارائه دهد(توضیحات تجربی وجود دارد ولی قطعی نیستند، البته تا الان) که چرا یک معماری کار میکند و معماری دیگر ناتوان است. البته ما در اینجا از مقالات و کتاب هایی که مهندسان باتجربه یادگیری ماشین ارائه داده اند، بهره خواهیم گرفت.
همانطور که گفتیم از convNet ها بهره میگیریم. و قصد دارم معماری ResNet را استفاده کنم.
کار خود را با ارائه توضیحاتی در مورد نمای چشم پرنده (bird’s eye view) آغاز میکنم، در واقع در مورد مفهوم MHR (modularity-hierarchy-reuse) برای یک معماری سیستم توضیح میدهم.

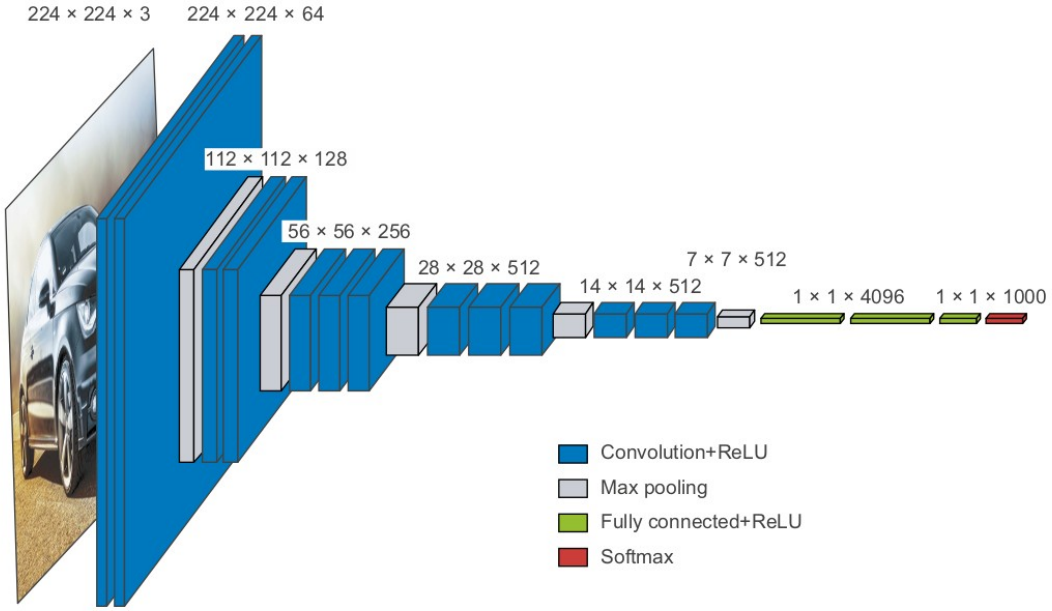
مفهوم MHR

اگر بخواهیم یک سیستم پیچیده را ساده تر کنیم، دستور جامعی که میتوانیم اجرا کنیم اینست که ابتدا سوپ های بی شکل از پیچیدگی(یک استعاره است) را در ماژول ها بریزیم. و ماژول ها را در یک سلسله مراتب مرتب میکنیم و سپس شروع به استفاده مجدد(reusing) یک ماژول در چندین مکان با توجه به نیاز ها میکنیم. این مفهوم کلی MHR است. این مفهوم قلب هر تشکیلات و سیستم پیچیده ای است، از بدن خود گرفته تا معماری ساختمان ها و سازماندهی ارتش و برنامه کامپیوتری بینایی ماشین.



یادگیری عمیق ، یک کاربرد از دستورالعمل(MHR) از بهینه سازی پیوسته توسط گرادیان کاهشی میباشد. به این صورت که ما تکنیک بهینه سازی کلاسیک(گرادیان کاهشی بر روی یک فضای تابع پیوسته) استفاده میکنیم و سپس به این فضای جستجو ساختار(ماژول بندی) میدهیم(لایه ها)، و سپس آنها را در یک سلسله مراتب(hierarchy) عمیق سازماندهی میکنیم، جایی که میتوانیم هر قسمتی از پردازش را

دوباره استفاده کنیم.(در شبکه های CNN اساسا اطلاعات را چندین بار و در چندین جا دوباره استفاده(reuse) میکنیم.)
ما این فرمول MHR را در تمام شبکه های cnn معروف میبینیم. آنها تنها در لایه ها ساختار بندی نشده اند، بلکه آنها در گروه های تکراری از لایه ها (که به آنها بلاک یا ماژول میگوییم) سازمان دهی شده اند.
به عنوان مثال معماری معروف VGG16 دارای بلاک های تکراری conv, conv, max pooling است.



سلسله مراتب های عمیق تر ذاتا بهتر است چراکه چندین بار استفاده مجدد از ویژگی ها(اطلاعات پردازش شده و نه خام) و در نتیجه انتزاع ها را انجام میدهند. به طورکلی، یک پشته از لایه های باریک دارای عملکرد بهتری از یک پشته سطحی از لایه های بزرگ است. البته باید ذکر کرد که عمیق بودن نیز دارای محدودیت است، و این محدودیت مربوط به مسئله ناپدید شدن گرادیان (vanishing gradients) است.
معماری های یادگیری عمیق بیشتر تکامل پیدا میکنند تا اینکه از قبل طراحی شوند. این به این معناست که ما از ساختار هایی استفاده میکنیم و سپس با تغییر برخی بلاک ها و ارتباطات آزمایش هایی انجام میدهیم و میزان موفق بودن هر ساختار و بلاک تعیین کننده استفاده از آن در ساختار نهایی مطلوب است.

توضیحات فنی و جزئی

بخشی از داده های خود را از سایت kaggle میگیریم و بخشی دیگر درکتابخانه های پایتون مثل keras موجود است.

و البته زبان برنامه نویسی python را انتخاب میکنیم چراکه برای پیاده سازی شبکه های عصبی بسیار کار ما را راحتتر میکند و کتابخانه های متنوع بسیار زیادی در خود دارد که مهم ترین آنها keras و tensorflow میباشد که محصول شرکت گوگل است و در کنار آن از کتابخانه های قدرتمند numpy , openCV بهره میگیریم. البته در کنار این کتابخانه ها از کتابخانه های دیگری نیز بهره برده ایم ولی اصلی ترین آنها، همین ۴ کتابخانه نام برده شده است.

در مرحله اول برنامه blur detection را مینویسیم چراکه شفاف بودن و شفاف نبودن به شدت تعیین کننده است. بهترین موتور های انجام دهنده ocr هم وقتی تصویر شفاف نیست، دچار اشتباه می شوند.

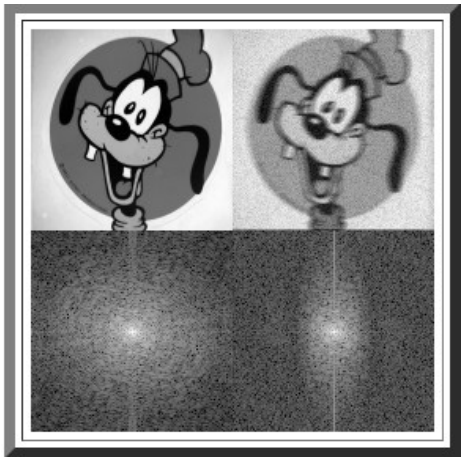
۲ روش اصلی برای انجام این تشخیص وجود دارد:

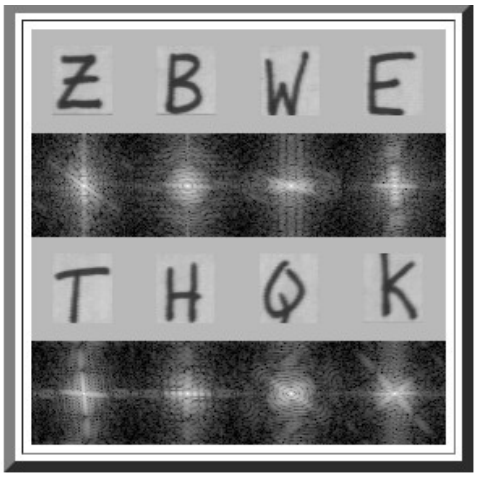
۱ ـ استفاده از واریانس عملگر لاپلاسین

۲ ـ استفاده از تبدیل فوریه سریع

روش اول به این صورت است که کرنل مخصوص اپراتور لاپلاسین(معمولا یک کرنل ۳×۳) را روی تصویر کانوال میکنیم و در نهایت واریانس درایه های ماتریس تصویر را محاسبه میکنیم. و همچنین باید گفت که محاسبه این عمل فقط با یک خط کد از کتابخانه openCv پیاده سازی میشود. ولی با این حال بنده از آن استفاده نکردم چرا که تعیین آستانه (threshold) برای تشخیص شفاف یا غیرشفاف بودن برای آن سخت است. (اگر شرایط مثل میزان روشنایی و ... کنترل شده باشد، به خوبی کار میکند.)

ولی استفاده از FFT بسیار جامع تر و فراگیر تر است. در این روش تبدیل فوریه تصاویر را بدست می آوریم. برای مثال به تبدیل فوریه های زیر توجه کنید:





در بینایی ماشین ، ما معمولا FFT به عنوان یک ابزار پردازش تصویر در نظر میگیریم که یک تصویر را در ۲ حوزه نشان میدهد:

۱. حوزه فوریه (حوزه فرکانس)
۲. حوزه s یا (Spatial Domain)

بنابراین FFT نشان دهنده یک تصویر در ۲ بخش حقیقی و موهومی است.
ما میتوانیم خیلی از کارهای پردازش تصویر مثل مات کردن ، تشخیص لبه ها و آستانه سازی(thresholding) و تجزیه و تحلیل بافت و در اینجا blur detection را با تحلیل و آنالیز کردن این مقادیر انجام دهیم.
بنده فایل مربوط به کاربرد تبدیل فوریه و حوزه فرکانس را که البته اسلاید مربوط به موسسه فناوری جورجیا است را ضمیمه گزارشکار کردم.
در برنامه مورد نظر برای انجام عمل fft از کتابخانه numpy بهره میگیریم . البته در کد مورد نظر با استفاده از کامنت گذاری جزئیات مربوطه را توضیح داده ام. ولی نکته ای که مهم است اینست که تعیین آستانه برای تصمیم گیری بسیار مهم است و اینکار تقریبا تجربی بدست می آید و مقدار default آن را ۱۵ قرار داده ام و ممکن است این مقدار در برخی از شرایط ، مقدار مناسبی نباشد و میتوان آن را با دادن آرگومان

python blur_detector.py --image image_name.jpg --thresh 10

تغییر داد ولی عدد ۱۵ در خیلی از مواقع درست عمل میکند و نیازی به تغییر نیست.
اگر تصویر شفاف بود به فرایند تشخیص ارقام ادامه میدهیم و اگر شفاف نبود برنامه را متوقف میکنیم و به کاربر میگوییم که تصویر بهتری ارائه دهد.
در مرحله بعد با استفاده از Guassian Blurring نویز را کاهش میدهیم. و سپس شروع به مرحله مهم لبه یابی میکنیم.
برای انجام این مرحله بنده از بهترین روش لبه یابی مورد استفاده در بینایی ماشین بهره میگیرم . نام این روش Canny edge detection است. بنده مقاله مربوط به آن را که در سال ۱۹۸۶ توسط John F.Canny ضمیمه گزارشکار کرده ام که میتوانید جزئیات ریاضی مربوط به آن را ببینید. ولی در اینجا بیشتر به مسائل کدزنی میپردازیم.
در پایتون با استفاده از کتابخانه openCv این وظیفه را انجام میدهیم. و بسیار ساده با استفاده از یک خط کد این وظیفه انجام میگیرد:

```
edged = cv2.Canny(blurred, 30, 150)
```

نکته ای که اینجا وجود دارد تعیین آرگومان های دوم و سوم است. که ما در کد مورد نظر از مقدار ۳۰ و ۱۵۰ استفاده کرده ایم. این دو آرگومان تعیین کننده تعیین کننده tresh هستند. و به طور کلی در این روش ۳ نوع treshold داریم:

- wide range
- mid range
- tight range

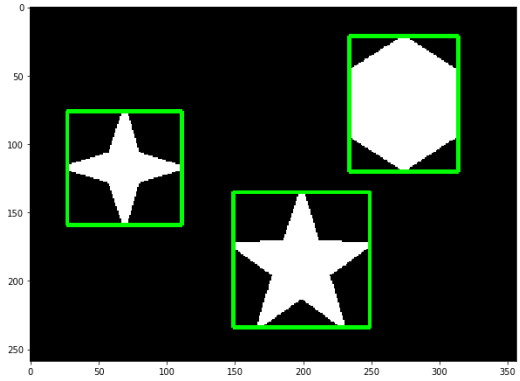
که مقادیر معمول مربوط به آن به صورت زیر است:

```
wide = cv2.Canny(blurred, 10, 200)
mid = cv2.Canny(blurred, 30, 150)
tight = cv2.Canny(blurred, 240, 250)
```

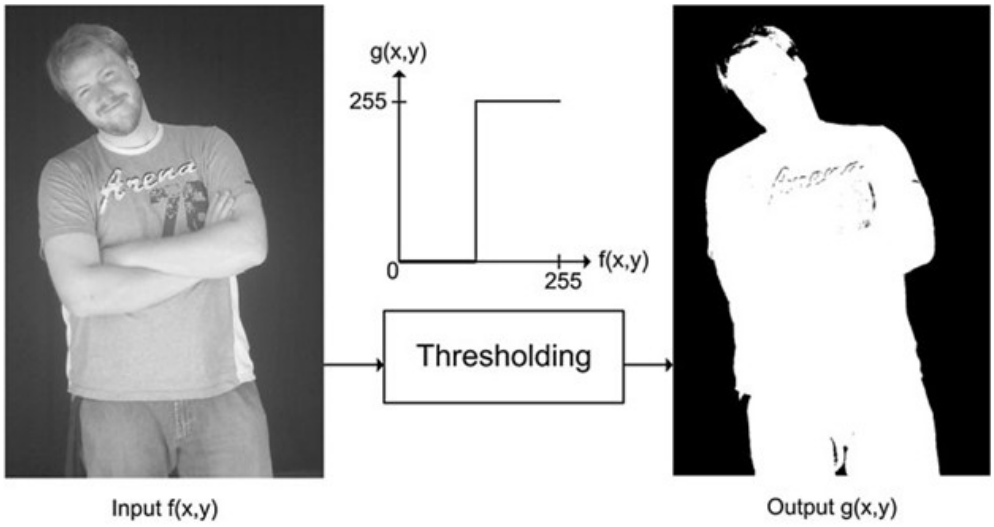
همانطور که میبینیم ما از mid استفاده کردیم، چراکه با توجه به همین تصویر که به عنوان مثال آورده شده است ، نشان دهنده برتر بودن به دیگر thresh ها است.

در مرحله بعد شروع به خطوط یابی با استفاده از دستور مفید findCountors که تابعی از کتابخانه openCV است ، میکنیم. توجه شود که در این مرحله، خطوط یابی را روی تصویری که از مرحله قبل بدست آمد انجام میدهیم.با اینکار خطوط سفید را انتخاب میکنیم. سپس اطلاعات مربوط به این خطوط را با استفاده از کتابخانه imutils در آرایه ای قرار میدهیم و دوباره با همین کتابخانه این اطلاعات را به ترتیب چپ به راست مرتب می کنیم.

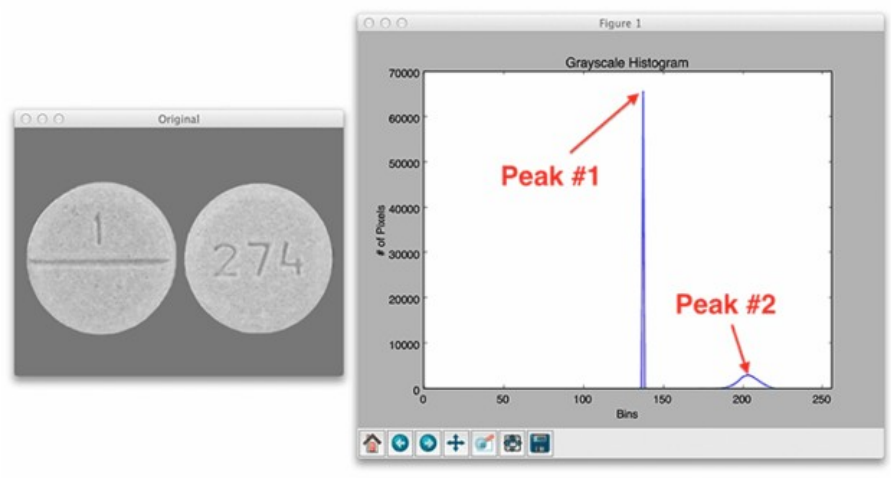
در مرحله بعد از همین اطلاعات استفاده میکنیم و با استفاده از دستور cv2.boundingRect تصویر را به صورت زیر محدوده سازی میکنیم و هر محدوده را جداگانه پردازش میکنیم. مثل تصویر زیر که البته مربوط به پروژه ما نیست.



در مرحله بعد thresholding را انجام میدهیم. در واقع thresholding تصاویر را باینری میکند. یعنی پس از این عمل یا هر پیکسل یا سیاه است یا سفید،به عبارت دقیق هر پیکسل یا مقدار ۰ را دارد یا ۲۵۵. همانطور که از اسم این عمل بر می آید ما به دنبال آستانه ای مثل T هستیم به این صورت که اعداد کوچک تر از T صفر شوند و اعداد بزرگ تر ۲۵۵ شوند. در تصویر زیر نمونه ای از این عمل را میبینیم.



تصویر بالا یک thresholding ساده است ولی ما در کد خود از otsu thresholding بهره برده ایم. این آستانه گذاری به این نکته توجه میکند که هر تصویر دارای شرایط نوری و کنتراست خود است، و نمی توان یک آستانه را تعیین کرد و سپس آن را روی هر تصویری اعمال کرد. روش اوتسو تصور میکند که هر تصویر دارای ۲ کلاس از پیکسل ها است : پس زمینه(background) و پیش زمینه(foreground).
سپس در نظر میگیرد که هیستوگرام مقیاس خاکستری(grayscale histogram) دوحالته است، یعنی اینکه دارای ۲ پیک می باشد.

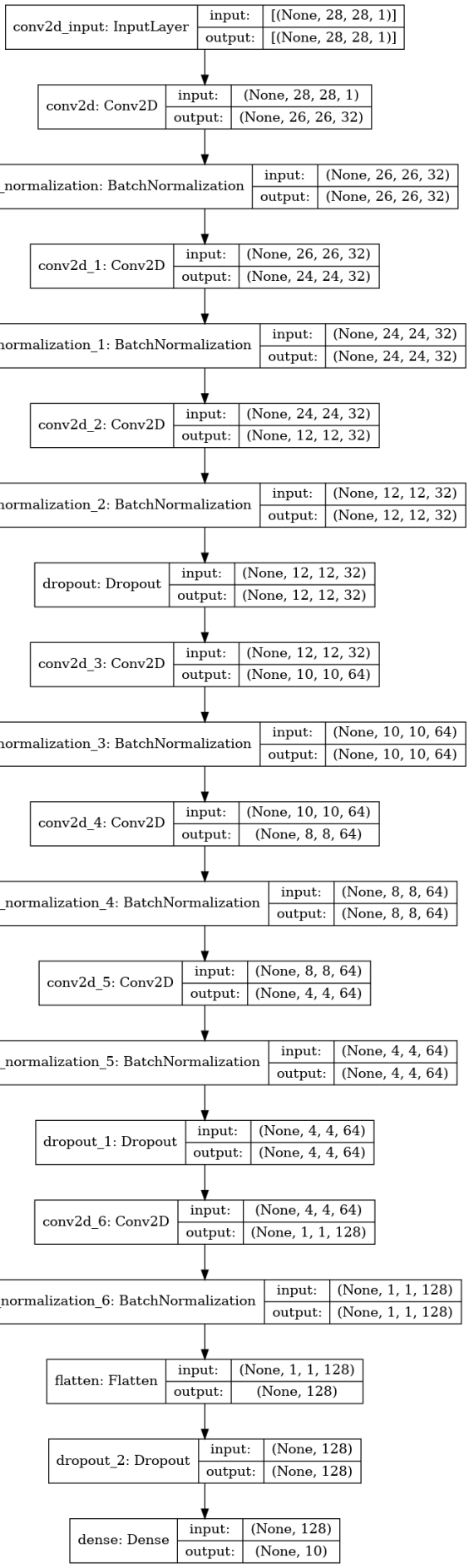


به نمودار بالا توجه کنید، در این نمودار پیک ۱ مربوط به رنگ یکنواخت پس زمینه می باشد. ولی پیک ۲ مربوط به رنگ قرص موجود در تصویر می باشد.
روش اوتسو آستانه T را به گونه ای محاسبه میکند که واریانس بین پیک پیش زمینه و پس زمینه کمینه شود.
ما این را تنها با استفاده از کد cv2.THRESH_OTSU پیاده سازی میکنیم.
این مسائل تنها پیش پردازش تصویر می باشد. بیایید دوباره مراحل را مرور کنیم:
ابتدا آمدیم تصویر را دریافت کرده و در صورت ۳ کاناله بودن (مثلا RGB) آنرا به مقیاس خاکستری بردیم که این به دلیل این بود که شبکه عصبی که در بخش بعدی به توضیح آن میپردازیم ، دارای شکل ورودی یک کاناله است که البته دلیل آن هم اینست که برای تشخیص اعداد، رنگی بودن یا خاکستری بودن تصاویر تاثیر گذار نیست فلذا هرچه میزان پردازش ها کمتر باشد برایمان به صرفه تر است پس مقیاس خاکستری را انتخاب میکنیم. سپس با استفاده از تبدیل فوریه سریع مشخص کردیم که آیا تصویر شفاف است یا غیر شفاف که اگر غیر شفاف باشد، برنامه قطع میشوند و به کاربر پیام میدهد که تصویر بهتری وارد کند. پس از این کار برای کاهش نویز از GuassianBlur استفاده کردیم . و پس از آن با روش canny لبه یابی را انجام دادیم. و سپس خطوط موجود در تصویر را با دستورات مربوطه (که در کد موجود است) ذخیره کرده و با توجه به آن bounding box ها را مشخص کرده و هر box یا مستطیل را به طور جداگانه پردازش خواهیم کرد. توجه کنید که هر یک از این مستطیل ها حداکثر یک رقم در درون خود خواهند داشت.(معمولا) مرحله بعدی ، بینایی ماشین است.

ـ بینایی ماشین و اطلاعات مربوط به پیاده سازی شبکه عصبی

در اینجا به مرحله تشخیص میرسیم. برای تشخیص از شبکه های عصبی cnn بهره میگیریم. که دلایل آن را در مقاله ذکر شده است. همانطور که در ابتدای این گزارشکار ذکر شد، انتخاب طراحی بیشتر یک هنر است تا علم. به همین دلیل اگر مقالات مربوط این حوزه را نگاه کنید میبینید که انتشار مقالات در این حوزه دارای سرعت زیادی است. هر محقق یک ساختار انتخاب کرده و روی آن آزمایشاتی انجام میدهد

و عملکرد و دقت آنرا ثبت میکند و در نهایت آنها را در یک مقاله منتشر میکند که اکثرا در arXiv موجود هستند. بنده نیز بی توجه به این مقالات نبوده و از ساختار های منتخب بهره برده ام.
مجموعه داده هایی که به عنوان داده های آموزشی برای تمرین دادن شبکه استفاده شده است، داده های MNIST میباشد که مخفف موسسه ملی استاندارها و تکنولوژی آمریکا میباشد. که این دیتابیس دارای تعداد بسیار زیادی از تصاویر ارقام میباشد. دلیل بهره گیری از آن اینست که این دیتابیس جامع و کامل است. و نکته جالبی هم که در این مورد وجود دارد اینست که اکثر معماری ها ، عملکرد و دقت خود را با این مجموعه داده آزمایش می کنند.
برای پیاده سازی شبکه عصبی از کتابخانه بسیار قدرتمند tensorflow که محصول شرکت گوگل است، بهره میگیریم. این کتابخانه به ما توانایی این را میدهد که هر نوع از شبکه ها را مدل کنیم. و سپس مدل مورد نظر را با استفاده از داده های آموزشی تمرین میدهیم.
بنده از ذکر جزئیات کد خود داری میکنم چراکه مستندات هر خط کد در سایت keras یا tensorflow موجود است.
در این پروژه ۲ مدل انتخابی داشته ایم که هر دو دارای دقت بالای ۹۵ درصد روی داده های تست هستند. مدل اول مدل ZFnet می باشد که با توجه به موفقیت آن در داده های مورد نظر انتخاب شده است. مدل دیگر اسم خاصی ندارد ولی دارای درصد دقت بیشتری است. به همین دلیل از مدل دوم به عنوان مدل نهایی خود استفاده کرده ایم. ساختار مدل دوم(که به عنوان مدل اصلی به کار گرفته شده) به شرح زیر است:


conv2d_input: InputLayer
input: [(None, 28, 28, 1)]
output: [(None, 28, 28, 1)]
conv2d: Conv2D
input: (None, 28, 28, 1)
output: (None, 26, 26, 32)
batch_normalization: BatchNormalization
input: (None, 26, 26, 32)
output: (None, 26, 26, 32)
conv2d_1: Conv2D
input: (None, 26, 26, 32)
output: (None, 24, 24, 32)
batch_normalization_1: BatchNormalization
input: (None, 24, 24, 32)
output: (None, 24, 24, 32)
conv2d_2: Conv2D
input: (None, 24, 24, 32)
output: (None, 12, 12, 32)
batch_normalization_2: BatchNormalization
input: (None, 12, 12, 32)
output: (None, 12, 12, 32)
dropout: Dropout
input: (None, 12, 12, 32)
output: (None, 12, 12, 32)
conv2d_3: Conv2D
input: (None, 12, 12, 32)
output: (None, 10, 10, 64)
batch_normalization_3: BatchNormalization
input: (None, 10, 10, 64)
output: (None, 10, 10, 64)
conv2d_4: Conv2D
input: (None, 10, 10, 64)
output: (None, 8, 8, 64)
batch_normalization_4: BatchNormalization
input: (None, 8, 8, 64)
output: (None, 8, 8, 64)
conv2d_5: Conv2D
input: (None, 8, 8, 64)
output: (None, 4, 4, 64)
batch_normalization_5: BatchNormalization
input: (None, 4, 4, 64)
output: (None, 4, 4, 64)
dropout_1: Dropout
input: (None, 4, 4, 64)
output: (None, 4, 4, 64)
conv2d_6: Conv2D
input: (None, 4, 4, 64)
output: (None, 1, 1, 128)
batch_normalization_6: BatchNormalization
input: (None, 1, 1, 128)
output: (None, 1, 1, 128)
flatten: Flatten
input: (None, 1, 1, 128)
output: (None, 128)
dropout_2: Dropout
input: (None, 128)
output: (None, 128)
dense: Dense
input: (None, 128)
output: (None, 10)

که البته فایل تصویر آن ضمیمه گزارشکار شده است. همچنین میتوان از ابزار قدرتمند tensorboard نیز بهره گرفت و نمودار مدل و نمودار پیشرفت و هیستوگرام را پس از تمرین شبکه بدست آورد ولی در اینجا لازم لازم نیست که ذکر شود.
همچنین اگر به نمودار بالا نگاه کنید میبینید که از چندین لایه ی کانولوشن استفاده کرده است و به همین دلیل به این شبکه cnn میگویند.
نکته ای که در مقاله آنرا ذکر نکرده ام ولی در اینجا استفاده شده است ، لایه ی batch normalization است. ترجمه فارسی آن میشود نرمال سازی دسته ای یک مرحله از فراعامل‌های γ و β که دسته‌ی {xi} را نرمال می‌کند در زیر آمده است.نماد μB و σ2B به میانگین و واریانس دسته‌ای که میخواهیم آن را اصلاح کنیم اشاره دارد که به صورت زیر است:

$$x_i \longleftarrow \gamma \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta$$

معمولا بعد از یک لایه‌ی تمام‌متصل یا لایه‌ی کانولوشنی و قبل از یک لایه‌ی غیرخطی اعمال می‌شود و امکان استفاده از نرخ یادگیری بالاتر را می‌دهد و همچنین باعث می‌شود که وابستگی شدید مدل به مقداردهی اولیه کاهش یابد.

پس از این مرحله نوبت به تنظیم پارامتر های کامپایل شبکه می رسد: optimizer شبکه را adam میگذاریم . این بهینه گر، الگوریتم آدام که در سال ۲۰۱۴ ارائه شد را پیاده سازی میکند. به طور خلاصه الگوریتم آدام همان روش گرادیان کاهشی استوکاستیک است که بر اساس تخمین تطبیقی(adaptive estimation) از moment های مرتبه اول و مرتبه دوم بنا شده است. (توجه: این moment با همتای آن در فیزیک مکانیک اشتباه گرفته نشود، این مفهوم مربوط به آمار و احتمال است)
در مرحله بعد تابع اتلاف(loss) خود را مشخص میکنیم. در کد مورد نظر از categorical cross entropy استفاده کرده ایم. در رگرسیون معمولی از میزان خطا به توان دو به عنوان تابع اتلاف استفاده میکنیم ولی در اینجا از آنتروپی متقاطع استفاده میکنیم. بررسی این مفهوم میتواند یک مقاله باشد ولی به طور کلی cross entropy loss به صورت زیر است:

$$\mathrm{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

در حالت categorical که در کد استفاده شده است از one-hot encoding استفاده میشود. با این کار مسئله مورد نظر به همان cross entropy معمولی تبدیل میشود.
نکته ی دیگری که باید گفت اینست که اگر پارامتر یادگیری را پس از هر epoch یا دوره تمرینی(که ما ۲۰ دور در نظر گرفتیم.) کاهش یابد یادگیری کمتر دچار پدیده فرابرازش میشود فلذا از LearningRateScheduler که تابعی از keras است بهره میگیریم و این موضوع را لحاظ میکنیم.

در مرحله بعد از ImageDataGenerator استفاده میکنیم. همانطور که از نامش پیداست این تابع با استفاده از چرخاندن ، کش دادن ، بالا و پایین بردن، تصاویر جدیدی ایجاد میکنند که مبادا شبکه ما دچار فرابرازش شود و به درستی یادنگیرد. با این کار شبکه ما دارای قابلیت تعمیم به تصاویر واقعی (real world) میشود.

و در نهایت شبکه را تمرین میدهیم.
برای تمرین دادن هر شبکه ی عصبی که با ۳ راه موجود است. راه اول استفاده از CPU ها که به طور پیش فرض در هر کامپیوتری فعال است و البته کم سرعت ترین روش است. راه دوم استفاده از GPU یا واحد پردازش گرافیکی است که توانایی پردازش موازش و همزمان چندین روال را میدهد و برای فعال سازی که معمولا محصول شرکت Nvidia میباشند و برای بهره گیری از این روش باید کامپیوتر تنظیم شود و Cuda روی آن نصب باشد و خب این روش دارای سرعت بسیار بیشتری از CPU است. روش سوم که سریعترین روش حال حاضر دنیا است ، استفاده از TPU یا Tensor Processing Unit می باشد که این ASIC محصول شرکت گوگل است و تنها برای تمرین دادن شبکه های عصبی که با tensorflow پیاده سازی شده اند، ساخته شده است. سرعت این قطعات ۱۰ برابر GPU است. مشخص است که در ایران این قطعات وجود ندارد ولی شرکت گوگل یک محیط اجرای مجازی برای آن ایجاد کرده است به نام colab که به تمام محققان هوش مصنوعی در سراسر دنیا توانایی بهره مندی از این قطعات را بدهد. البته این محیط مجازی دارای هر ۳ حالت cpu , gpu, tpu میباشد و خود کاربر باید آن را مشخص کند.
همچنین باید گفت که استفاده از tpu کار چندان آسانی نیست، به این صورت که اول باید همان محیط مجازی با چند خط کد configure شود و سپس نکته به شدت محدود کننده اینست که لود کردن دیتاست های بزرگ تنها با استفاده از google cloud صورت میگیرد(که برای ایرانیان مسدود است) و نمیتوان از جای دیگری آن ها را لود کرد، یا این باید ابتدا تمام داده ها را در RAM لود کرد که آن هم دارای محدودیت است چرا که رم حداکثر ۱۲ گیگ ظرفیت دارد.(برای ظرفیت بیشتر باید پول پرداخت)

از تمام این مسائل جزئی که بگذریم داده ها را با استفاده از هر کدام از این سه روش پردازش میکنیم و در نهایت شبکه ی نهایی(که تمرین داده شده است) را ذخیره میکنیم.(با پسوند h5)

در مرحله بعد این فایل دارای پسوند h5 را در محل اجرای برنامه کپی میکنیم و هر موقع که نیاز به انجام فرایند تشخیص داشتیم ، این فایل را با load_model که تابعی از tensorflow است، لود میکنیم و تشخیص را با استفاده از model.predict انجام میدهیم.
سپس با cv2 روی تصویر تشخیص مورد نظر را به نمایش در می آوریم.